Instituto Tecnológico de Costa Rica

Segundo Avance: Definición Intérprete

Integrantes:

Josué Arce González

Daniel Montero Carvajal

Curso de Compiladores e Interpretes

01/06/2018

Contenido

[Introducción 3](#_Toc515570418)

[Soluciones e Implementación 3](#_Toc515570419)

[Resultados Obtenidos 3](#_Toc515570420)

[Manual de Pruebas 3](#_Toc515570421)

[Conclusiones 3](#_Toc515570422)

[Bibliografía 3](#_Toc515570423)

# Introducción

Interpretar instrucciones es un proceso arduo, elaborado, el cual requiere de mucho análisis, ya que se deben de tener en cuenta todos los posibles casos que se puedan dar, evaluar la sintaxis escrita, verificar si cumple con lo solicitado y mostrarle un resultado al respecto al usuario; de esto trata el siguiente documento, hace énfasis a las pruebas hechas sobre el intérprete generado, se muestran los resultados obtenidos, secciones que no se cumplieron y porqué, además, se adjuntan las soluciones tomadas y su respectiva explicación, por último, un pequeño comentario que refleja nuestras opiniones luego de haber culminado con el susodicho avance.

# Soluciones e Implementación

A continuación, se detallan los elementos utilizados que fueron la solución a la etapa #3:

1. EvaluationStack:
   1. Encargado de almacenar durante un período todos los resultados, valores relacionados a una variable, función, entre otros, para así manipular de los mismos y generar una respuesta, se dice que contendrá los datos de manera periódica, porque estos van a estar constantemente en cambio, es decir, se introduce un valor, utilizando el método ***pushValue***, el cual recibe un valor de cualquier tipo y se inserta en la pila,pero este luego será obtenido para aplicarle a este un cálculo matemático, y luego se inserta el nuevo valor de nuevo a la pila; por otro lado, también existe un método llamado ***popValue*** el cual permite obtener el valor que se encuentre en el “tope” de la pila.
2. Interpreter:
   1. Encargado de generar todos los resultados a partir de una, dos o más variables, métodos, entre otros, dependiendo del cálculo solicitado este va realizar el proceso necesario para así poder generar una respuesta (error o valor) a partir de lo solicitado; cada método creado en esta sección tiene un propósito importante, hay métodos que están enlazados, es decir, uno depende del resultado de otro, por lo que fue importante al inicio pensar en una estrategia para que no hubiesen atrasados o errores conforme se desarrollaba el programa o a futuro.
   2. Conforme se van visitando los métodos necesarios para fabricar el resultado, se van realizando al mismo las validaciones necesarias para que el proceso no llegue a fallar, se debe siempre mostrar un resultado al usuario; por cada método se obtienen el primer, los 2 primeros, o más valores que se encuentren en el “tope” de la pila, claro está que al inicio de programa se debe establecer una instancia de la clase ***evaluationStack***¸ para así poder acceder a los métodos dentro de esta, entonces conforme se vayan necesitando de valores se extraen de la pila, se les aplica una fórmula matemática y el paso siguiente es insertar el resultado de dicho calculo en la misma pila, ya que se va necesitar luego tanto para otro calculo o para mostrarlo en pantalla.
   3. Cabe destacar un importante detalle, cuando se tiene una función dentro de una lista, la única forma de poder acceder a ella es asignando la posición de la lista a una nueva variable, luego utilizando la nueva variable es posible ejecutar la función creada en la lista, por ejemplo: let lista = [fn(x){puts(a+a)}] – let funcion = lista[0] – funcion (5), como se puede apreciar, esta es la única manera de poder ejecutar una función ubicada dentro de una lista.
3. DataStorage:
   1. Es la base de esta tercera etapa, ya que toda la información ingresada en el IDE, se va almacenar en esta sección, absolutamente todo se almacena, con el siguiente orden: **nombre de la variable -> valor asignado en el IDE**; esto permite poder tener el identificador junto a su respectivo valor, entonces su principal uso es el poder acceder a un identificador en específico, para esto se elaboraron dos métodos, los cuales son llamados ***getData***, uno recibe la posición exacta de lista donde están concentrados los datos, y el segundo método recibe el nombre del identificador, para así buscarlo a partir del respectivo nombre asignado; otro método importante es el ***cleanData***, este nos permite eliminar todos los registros que se encuentren actualmente dentro de la lista, entonces en pocas palabras lo que realiza es vaciar completamente la lista. Pero no podemos dejar de lado el método más importante de esta sección, y es la función llamada ***addData***, este método recibe 3 parámetros, el nombre asignado, el valor establecido para dicho identificador y la posición actual de la lista, entonces se introduce el nuevo registro en la posición recibida + 1, esto para asignarla en la siguiente posición de la lista enlazada.

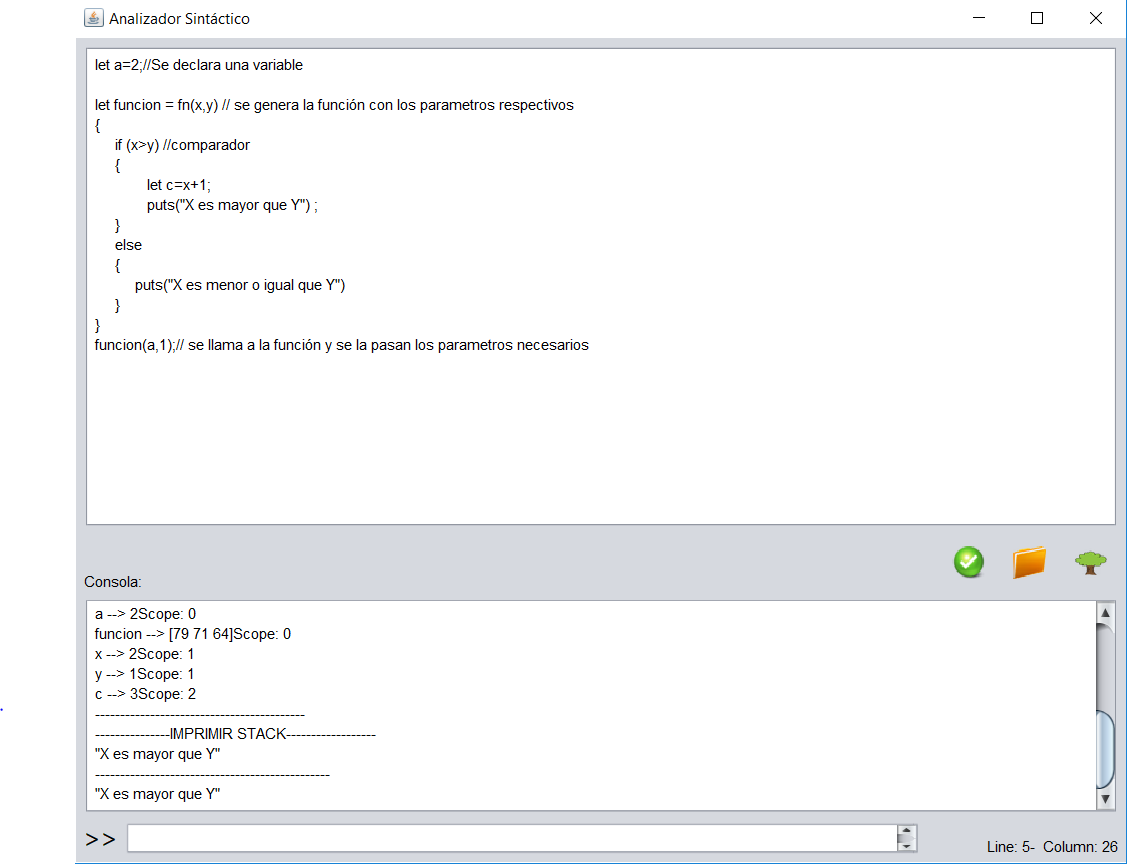
# Resultados Obtenidos

|  |  |  |
| --- | --- | --- |
| Aspecto Por Evaluar | Estado (0%-100%) | Justificación en caso de no estar el aspecto al 100% |
| DataStorage | Se encuentra funcionando en perfectas condiciones, introduce y almacena cualquier tipo de dato sin problemas. | N/A |
| EvaluationStack | Se encuentra funcionando en perfectas condiciones, almacena y permite introducir y acceder a todos los valores introducidos. | N/A |
| Interprete | Se encuentra funcionando en perfectas condiciones excepto por el proceso de recursión en las funciones, todos los otros casos posibles fueron cubiertos, cualquier tipo de función matemática, consulta deseada retorna un valor. | La recursión no pudo ser implementada por falta de tiempo, desarrollarlo toma tiempo para pensar en una estrategia y para generarla se necesita más tiempo de igual manera. |
| Estado General | El avance # 3 fue abarcado correctamente dentro del ámbito de las pruebas realizadas. | N/A |

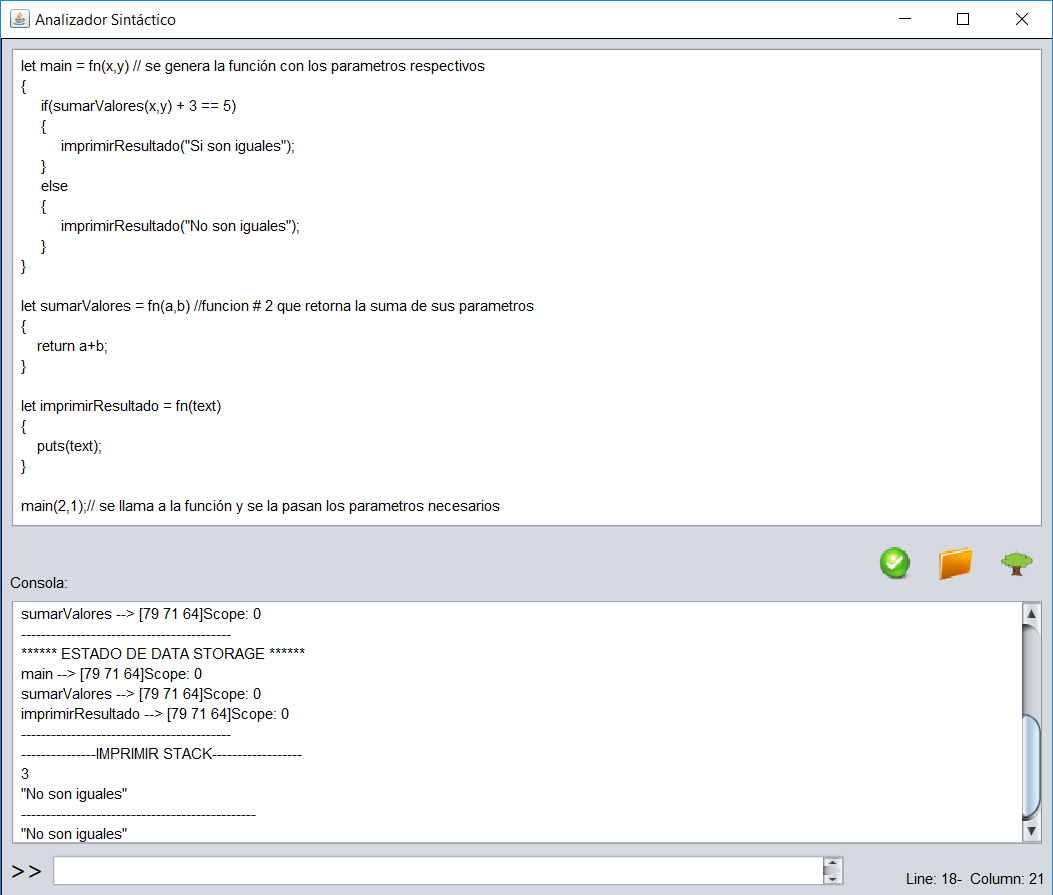
# Manual de Pruebas

Se van a dividir las pruebas por tipos, es decir, funciones, operaciones aritméticas y así sucesivamente.

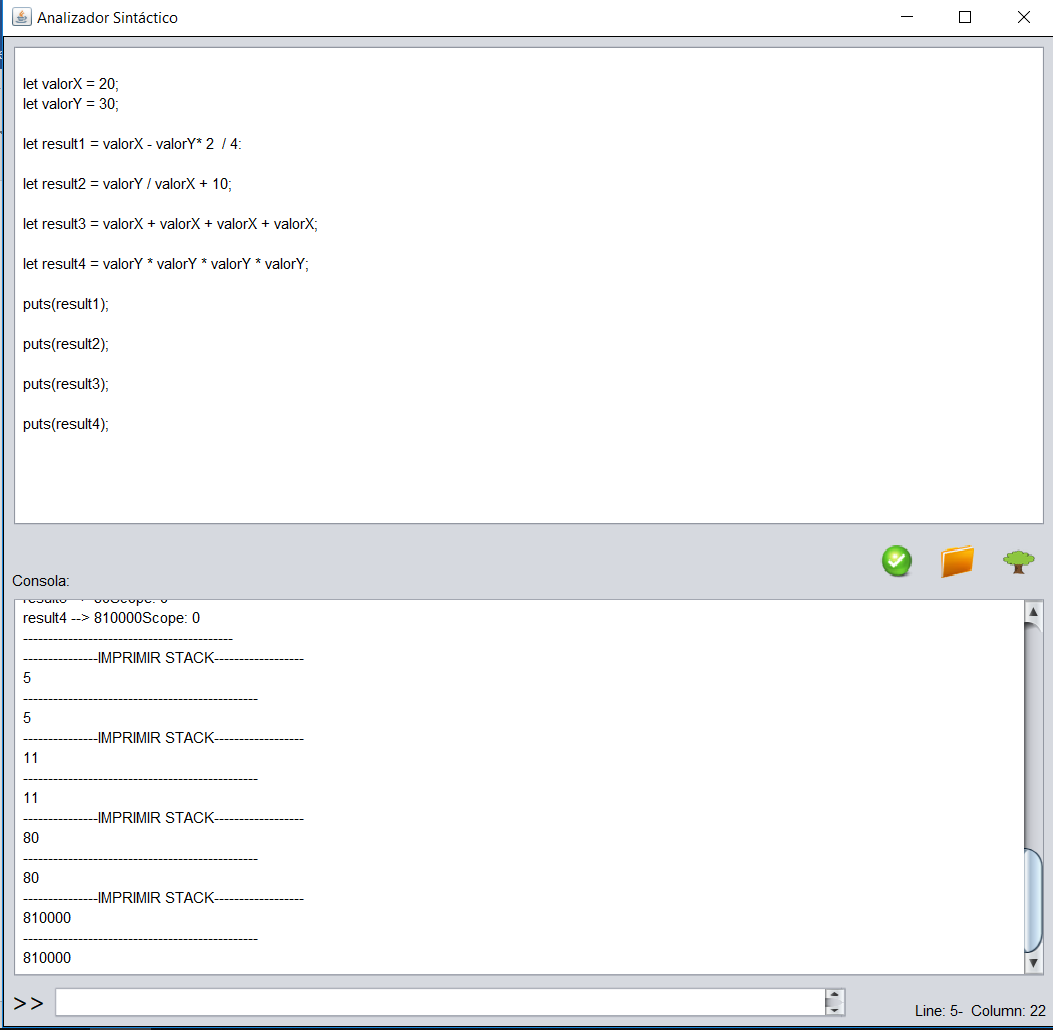
1. Funciones:
   1. En esta primera ejecución se pone a prueba los **parámetros**, la **función** y el uso del **IF**, como se puede apreciar, la función utiliza los parámetros de forma correcta, una forma de comprobarlo es en el IF, ya que como en este caso se le está enviando una variable con el valor 2 y otro valor 1, por lo que si se compara que 2 es mayor que 1 da como resultado verdadero, por lo que lo siguiente será ingresar al “***then”*** del IF, luego se genera otra variable para afirmar el uso de variables dentro de funciones, por último se imprime usando la función  ***puts*** un resultado que afirme la comparación anterior.

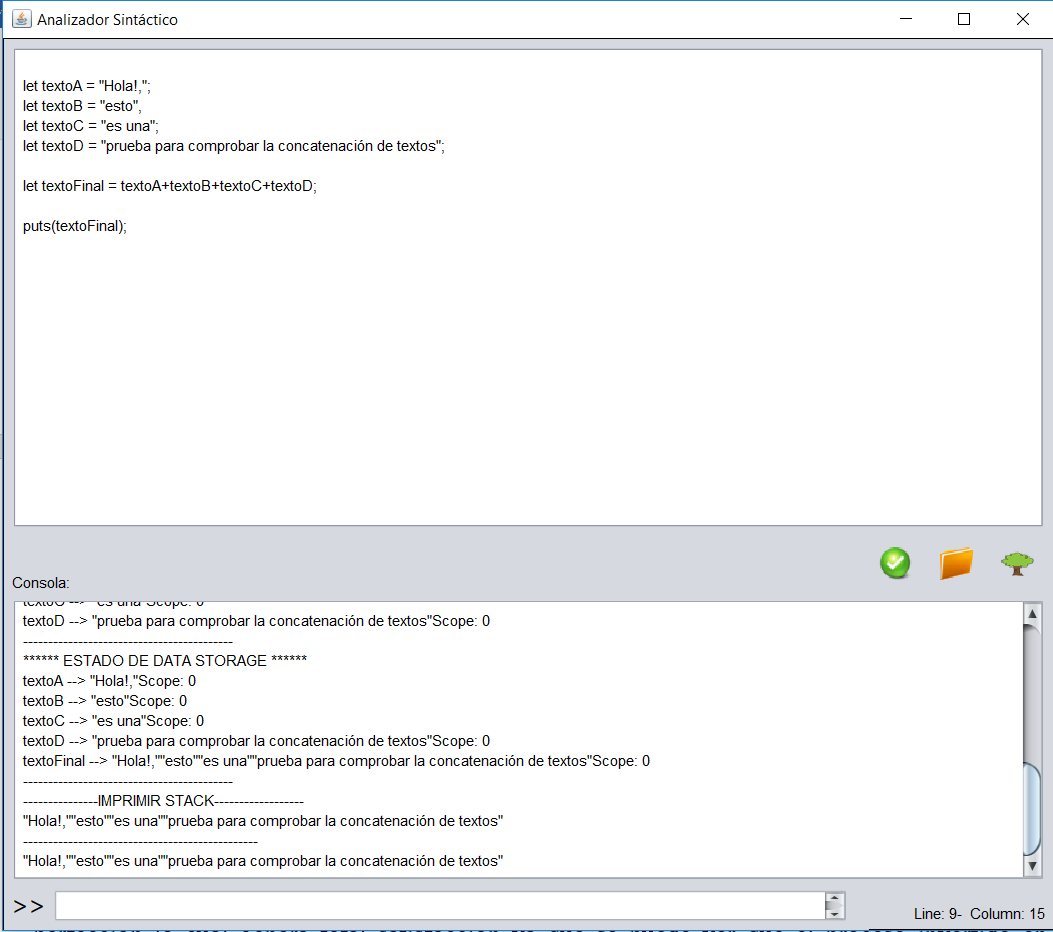


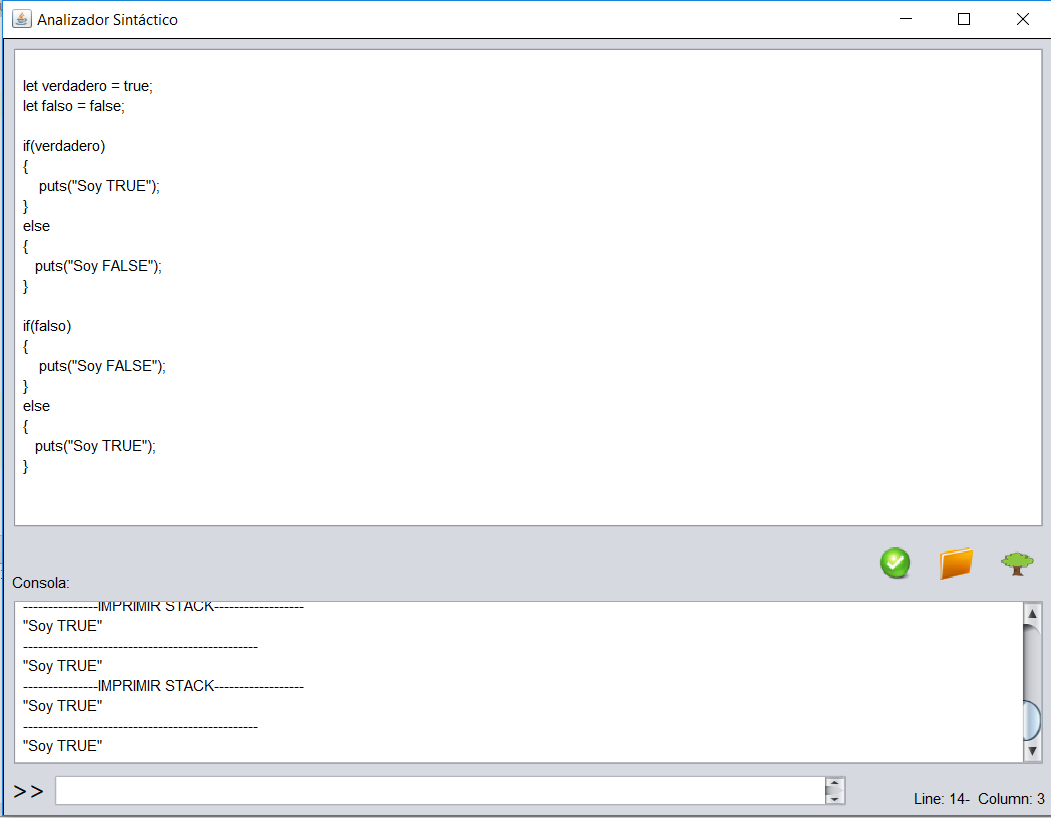
* 1. Para este caso se elaboraron 3 funciones, **main, sumarValores, imprimirResultado,** la función main como su nombre dice es la principal, esta recibe 2 valores por parámetro, se encarga de llamar a la función sumarValores, esto se realiza dentro de un comparador **IF**, en donde se le pasan los parámetros recibidos en el main a la función sumarValores para que esta retorne la suma de los mismos, luego, una vez obtenidos el resultado de la función se le agrega un **+3**, y seguidamente se compara esta suma con **5**, en caso de que la comparación sea cierta este va entrar en el primer **blockStatement,** como se puede apreciar en la consola, el resultado es “No son iguales”, esto se puede encontrar en el segundo **blockStatement** del comparador IF, dentro de cada blockStatement, se encuentra una llamada a la última función mencionada anteriormente, **imprimirResultado**, el cual recibe un texto por parámetro y lo imprime utilizando la función **puts**, entonces, en este caso el texto enviado es “No son iguales” ya que la comparación es falsa, (2+1) + 3, esto es igual a 6 por lo que no es igual a 5.

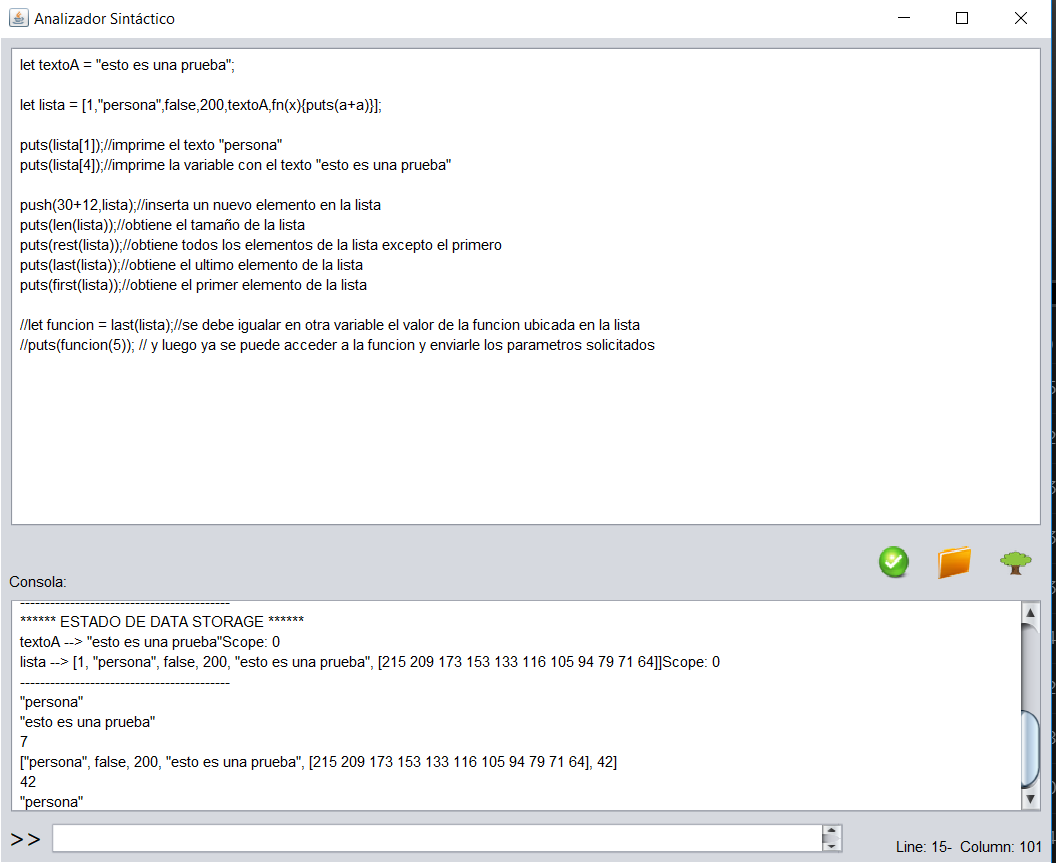


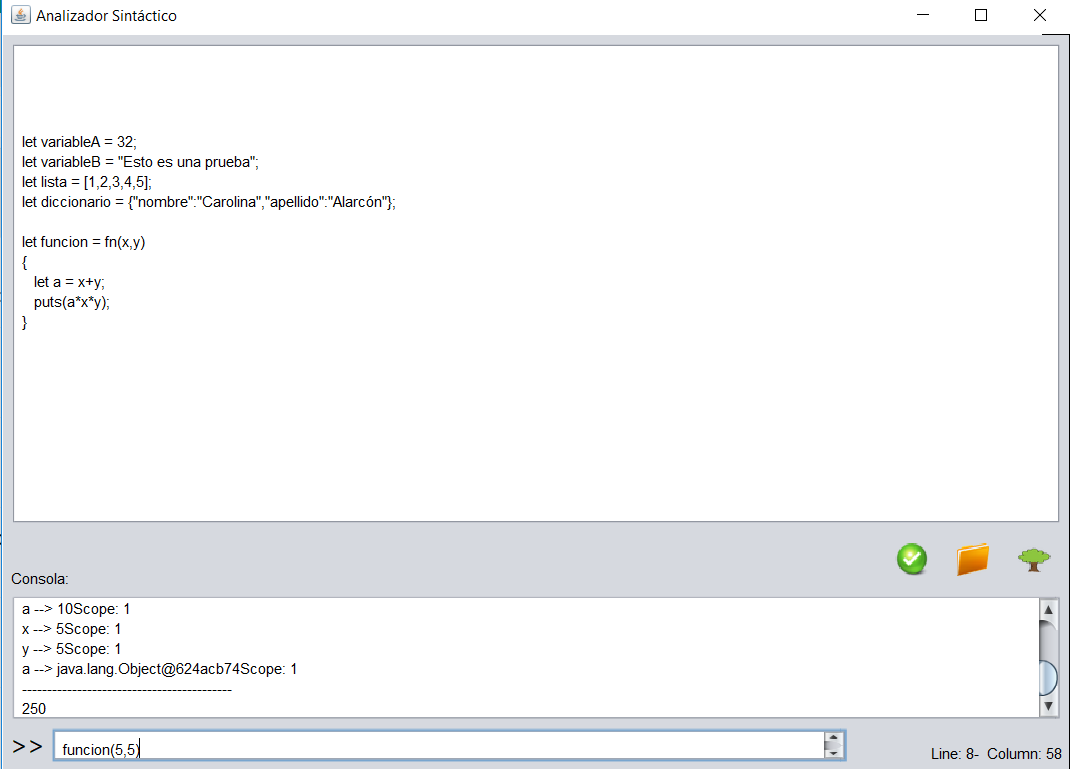
1. Operaciones aritméticas:
   1. En esta primera prueba de operaciones aritméticas, se ponen en juego las 4 principales formas de operar valores numéricos, sumar, restar, multiplicación, división, se tienen 2 variables, valorX con el valor de 20 y valorY con el valor de 30, luego se crean 4 variables más, result1, result2,result3, result4, cada una de estas variables contiene una serie de operaciones, por ejemplo, result3 posee 4 sumas, se suma valorX 4 veces, el resultado de esto es 80, ya que 20 + 20 + 20 + 20 = 80, esto se puede comprobar en la consola, otro ejemplo es el result2, el cual divide primero valorY entre valorX y luego se le suma 10, entonces tenemos que 30 / 20 + 10 = 11,5 pero se elimina la decimal y se mantiene el numero entero, el cual es 11, esto se aprecia también en la consola.



1. Concatenar textos:
   1. Sumar cadenas de caracteres es básicamente juntar un texto con otro en una fuente, es decir, podemos decir que concatenar es tener “Hola soy” y “Estudiante TEC”, y que podamos tener luego “Hola soy Estudiante TEC”, esta es la prueba que se realizó a continuación, se tienen 4 variables que almacenan 4 textos diferentes, y una variable llamada textoFinal la cual almacena la concatenación de las 4 anteriores, hay que dejar claro que esto solo funciona con el operador  **+,** cualquier otro operador utilizado provocará que muestre un error de sintaxis en la consola, entonces, teniendo la suma de lo anterior se genera un resultado el cual es “Hola!, esto es una prueba para comprobar la concatenación de textos”, este texto se imprime en consola para comprobar su validez.
2. Valores booleanos:
   1. En este caso lo que se hará es poner en práctica los valores booleanos, **true y false,** se tienen 2 variables, verdadero es de tipo true y falso es de tipo false, luego se generan 2 comparadores IF, el primero compara el valor de verdadero, el cual es true por lo que entrará al primer blockStatement, y por ende se imprime el texto “Soy TRUE”, en el segundo IF se compara el valor de falso, el cual es false, por lo que entrará al segundo blockStatement, y se imprime el texto “Soy TRUE” también, ya que lo que se está consultando es si es falso es verdadero, lo cual no es cierto, entonces por ende entra al segundo blockStatement, estos textos se pueden verificar en la consola.



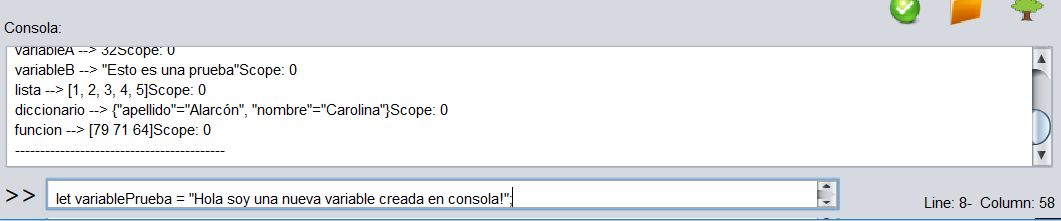
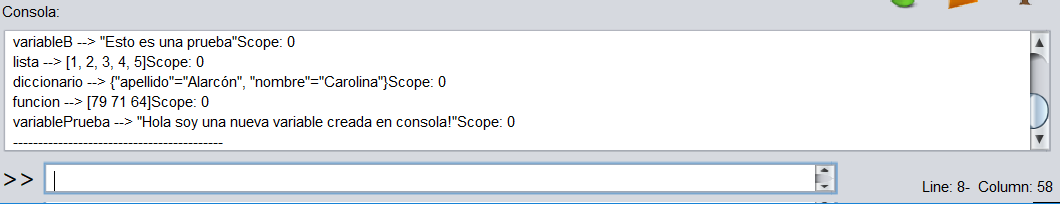
1. Hash y Arrays:
   1. Listas(arrays):
      1. En la siguiente imagen se muestran las pruebas realizadas a listas, se ponen a prueba todas sus funciones, tales como acceder a un elemento de la lista, push, first, last, rest, len, estas últimas permiten obtener elementos de la lista, la lista puede almacenar cualquier tipo de dato, pero hay un caso en particular, las funciones, la lista puede almacenar una función pero para acceder a esta función se requiere de trasladar la función localizada en la lista a otra variable(como en el ejemplo), para poder así manipular la función.
   2. Hash(diccionarios):
2. Escribir en consola:
   1. Un aspecto importante en este avance es el hecho de poder llamar funciones, imprimir y definir nuevas variables, entonces a continuación se muestran unas pruebas realizadas para demostrar lo anterior.



Como se puede apreciar, en la consola, se debe escribir una función, puts, o declarar una nueva variable, en este caso se está llamando una función ya creada, a esta se le pasan los parámetros necesarios, como se puede observar en la imagen de la derecha el valor **250** es el resultado de la función llamada.

# 

En este caso, lo que se está probando es declarar variables en la consola, el ejemplo muestra la declaración de una nueva variable llamada **variable Prueba,** en la imagen de la derecha, se observa que se agregó con éxito la nueva variable con su respectivo valor asignado.



# Conclusiones

Exitosamente se lograron abarcar los puntos establecidos al inicio, el programa realiza lo solicitado por el usuario y siempre muestra un resultado al respecto, el intérprete funciona a la perfección lo cual genera total satisfacción ya que se puede ver que el proceso invertido en funcionamiento; luego de haber experimentado toda la creación de un sistema compilador, se puede responder con seguridad, que no es nada fácil, es para nada sencillo generar un programa que permita leer un código y mostrar un resultado, es un largo y complejo proceso, el cual debe manejarse con sumo cuidado y abarcando todas las etapas, cabe destacar que en el desarrollo del mismo solo se cubrieron 3 etapas, pero faltaron muchas más, por ejemplo, el generar un archivo con el código fuente, además, de que el lenguaje ***Java*** te asiste de gran manera, te brinda muchas opciones al momento de programar, por lo que resulta de una u otra manera un peso menos.

Es increíble ver todos esos compiladores ya creados, que están en funcionamiento, y ahora se puede concluir el respeto y sobre todo valorar el trabajo dedicado por los desarrolladores de código, ya que gracias a ellos tenemos la dicha de poder escribir programas. El intérprete para terminar es una base fundamental a tener en cuenta, dentro de este ocurre la magia, es decir, es aquí donde las operaciones aritméticas, comparaciones, llamadas a métodos se deben de evaluar a la perfección, el susodicho genera la respuesta final para el usuario.

# Bibliografía

N/A